# THE ZX
# ASSEMBLER & DISASSEMBLER



Bob Berch

Rochester NY 14620

Contents



Appendicies

## I. Introduction

Welcome to the ZX Assembler/Disassembler by Bob Berch. The primary purpose of this program is to help you write and debug machine code programs. Included in this one program is an assembler utilizing standard Z80 mnemonics, a disassembler giving hex or decimal output to the screen or printer, a hex or decimal memory editor, a REM generator to make REM statements to save machine code in, and a set of binary SAVE and LOAD routines. The program is a little over 4K long and is supplied on cassette in two versions, back to back:

1. Hi-mem, above RAMTOP, program at 28-32K. Those with only a 16K rampack (the minimum configuration) must use this one.
2. Lo-mem, in the ROM transparent area, program at 12-16K. To use this version you must have memory in this area as with a 64K Rampack or a Hunter board.

In addition, this program is available on EPROM in the lo-mem version, without the LOAD and SAVE routines, as an exactly 4K program. A plug-in cartridge kit is available to hold the EPROM.

If you can use the lo-mem version,it is recomended you do so as you will find it more out-of-the-way and also more immune to crashes from your generated machine code gone haywire.

This manual will not attempt to teach you machine code programming and you are refered to some of the many Z80 books on the market in appendix D . You are given sample programs in part IV and a fairly complete list of usable ROM routines and their calling sequences in appendix C and a similar list of usable Assembler/Disassembler routines in appendix B.

To load the tape, choose the proper version,and use the name 'A3MB'. It will auto-boot itself in place and the hi-mem version will lower RAMTOP. A title page will be displayed.

Assembly language source code for the assembler is entered under Basic in REM statements; the specifications are given below. The Assembler/Disassembler is called with a RAND USR command and the assembly is started by pressing the Z key. If no errors are found, the object code is deposited in a string named A$. Here it may be viewed with the disassembler, moved to another part of memory or a REM statement, edited, or saved as a binary file.

The RAND USR command for each version is:

```
     low-mem      RAND USR 13616
     hi-mem       RAND USR 30000
```

Once the RAND USR call has been made, the various functions of the Assembler/Disassembler are called with single letter key commands (letters A to Z). An asterisk in the lower left of the screen signifies a wait for one of these commands.

## II. The Key Commands

**First, observe the four display modes illustrated below:**

### Disassembly Mode

**Hex**

```
0000 ▮D3 FD       OUT FD ,A
0002  01 FF 7F    LD BC,7FFF
0005  C3 CB 03    JP 03CB
0008  2A 16 40    LD HL,(4016)
000B  22 18 40    LD (4018),HL
000E  18 46       JR 0056
0010  A7          AND A
0011  C2 F1 07    JP NZ,07F1
0014  C3 F5 07    JP 07F5
0017  FF          RST 38
0018  2A 16 40    LD HL,(4016)
001B  7E          LD A,(HL)
001C  A7          AND A
001D  C0          RET NZ
001E  00          NOP
001F  00          NOP
0020  CD 49 00    CALL 0049
0023  18 F7       JR 001C
0025  FF          RST 38
0026  FF          RST 38
0027  FF          RST 38
0028  C3 9D 19    JP 199D
```

**Decimal**

```
0   ▮211 253      OUT 253 ,A
2    1 255 127    LD BC,32767
5    195 203 3    JP 971
8    42 22 64     LD HL,(16406)
11   34 24 64     LD (16408),HL
14   24 70        JR 86
16   167          AND A
17   194 241 7    JP NZ,2033
20   195 245 7    JP 2037
23   255          RST 56
24   42 22 64     LD HL,(16406)
27   126          LD A,(HL)
28   167          AND A
29   192          RET NZ
30   0            NOP
31   0            NOP
32   205 73 0     CALL 73
35   24 247       JR 28
37   255          RST 56
38   255          RST 56
39   255          RST 56
40   195 157 25   JP 6557
```

### Data Mode

**Hex**

```
0000 ▮  D3   PEEK     211  0
0001    FD    CLEAR   253  1
0002    01   ▪        1    2
0003    FF    COPY    255  3
0004    7F   ?        127  4
0005    C3   ?        195  5
0006    CB   ACS      203  6
0007    03   ▬        3    7
0008    2A   E        42   8
0009    16   -        22   9
000A    40   RND      64   10
000B    22   6        34   11
000C    18   /        24   12
000D    40   RND      64   13
000E    18   /        24   14
000F    46   ?        70   15
0010    A7   ▣        167  16
0011    C2   TAB      194  17
0012    F1    LET     241  18
0013    07   ▛        7    19
0014    C3   ?        195  20
0015    F5    PRINT   245  21
```

**Decimal**

```
0  ▮  211  PEEK     D3  0000
1     253   CLEAR   FD  0001
2     1    ▪        01  0002
3     255   COPY    FF  0003
4     127  ?        7F  0004
5     195  ?        C3  0005
6     203  ACS      CB  0006
7     3    ▬        03  0007
8     42   E        2A  0008
9     22   -        16  0009
10    64   RND      40  000A
11    34   6        22  000B
12    24   /        18  000C
13    64   RND      40  000D
14    24   /        18  000E
15    70   ?        46  000F
16    167  ▣        A7  0010
17    194  TAB      C2  0011
18    241   LET     F1  0012
19    7    ▛        07  0013
20    195  ?        C3  0014
21    245   PRINT   F5  0015
```

In disassembly mode, the leftmost column of numbers is the address, the raw data at those addresses is at center, and the assembly mnemonics are at the right.

In data mode there are five columns: the leftmost and rightmost columns are the addresses (in hex and decimal), next in is the data at each address (also in hex or decimal), and finally in the center is the Sinclair character equivalent of that data.

Now, the key commands in detail:

**A** enter address. A > will appear ( or a >> if you are in hex mode) and you enter the digits for the address for the top of the page (TOP). Hex addresses are always four digits and need no return. Decimal addresses always need a return. The delete key deletes the entire entry and you start again.

**B** looks backward in memory. If you're in data mode, you go backward 22 addresses. If you're in disassembly mode, you go back 30 addresses. Disassembly backpages will generally overlap, but occasionally will not, as with a field of all NOP's. In addition, disassembly backpages will often land in the middle of a multi-byte instruction and the first instruction or two will be gibberish. Just be aware of it.

**C** continue looking foward in memory. In both data and disassembly mode the top of the new screen starts with the byte after the last byte on the present screen.

**D** data/disassembly switch. Changes from disassembly to data mode and back again.

**E** edit memory. If in disassembly mode, you will be changed to data mode. A cursor will appear at the top address and an input prompt (hex or decimal, depending on what mode you were in last) at the bottom left. Hex data is two digits and needs no return; decimal data always needs a return. Entered data will be poked into memory at the cursor address and the cursor will be moved down the page. The up and down arrow keys can be used to move the cursor over any addresses.

**F** set the EOF address, that is the end of file address. This address is used by the L, M, S, T, and X commands.

**G** generate a REM. Enter the number of spaces desired within the REM. Makes a 1 REM filled with x's.

**H** hex/decimal switch. Changes from hex to decimal mode and back again.

L load binary. Use the A command to set the first address to be loaded and the F command to set the last. A load will procede until the last byte is loaded or the break key is pressed.

M move memory. Moves a block of memory between and including The TOP address ([<] cursor) and the EOF address to the address entered on the input prompt. The EOF address must be greater than the TOP address. If the block of memory overlaps its future position, this routine intelligently avoids overwriting it by choosing a top-down or bottom-up move.

Q (shifted) quit. Used to exit any input prompt to get the asterisk or to leave the asterisk to return to Basic.

S save binary. This is the complement of L, of course. Use the A anf F commands to set the limits to save; you save up to and including the EOF byte.

Besides being used to to save files of your assembled code, these commands can be used to load and list 'unlistable' programs or load, edit, and save damaged tapes. No name or header is saved with these files, just raw data.

During a save, pressing break will return to Basic.

T type output. Outputs to a Timex or Sinclair printer, in whatever mode you are in, the present and consecutive screens up to the EOF address. Like save, you may break, but will be returned to Basic.

X clear. Sets to zeros all memory from the TOP address to and including the EOF address.

Z assemble. Read on.

Note: some commands ask OK? before proceding. A 'Y' or an enter will procede. A 'N', a 'break' or a shifted Q will allow you an escape.

III. The Assembler

Before calling the assembler (with the Z command), the source code must be entered and present in memory. This is done under Basic, entering each statement in a REM statement. Here are the rules:

1. The operation code must be immediatly after the REM followed by one or more blanks or an end of line. The operand follows, parts of the operand seperated by a comma, then an optional semi-colon and comments. Blanks are ignored in the operand but must not appear in the op code. A line full of comments starts with a semicolon.

Your REM statements are edited or deleted like any Basic statement.

2. In the operand, variables in the Basic variable area may be used in place of any number. Variables so defined must be at least three characters long and not start with the letter H. Define the variables with LET statements, either in immediate mode or as program statements after a END statement (see below) The second way better documents your program. Remember to 'RUN' your program before calling the assembler to execute your LETs.

3. Hexadecimal constants are preceded by an 'H' and are two, three, or four digits long. When only two digits are given for a 16 bit constant, they will be the low order byte.

Decimal data is accepted in the range zero to 65535.

Character data within quotes is allowed, one or two characters long. More than two characters may be given, but only the last two are recognized. Any character code is allowed except the quote (0Bh) and the enter (76h).

examples:

| valid constants | | invalid constants |
|---|---|---|
| H543 | (0543h) | +1 |
| '=' | (14h or 0014h) | 70000 |
| 8193 | (2001h) | H7 |
| 'WXYZ' | (3E3Fh) | -1 |

4. Branch instructions use the line number or a variable equal to a line number to reference an address, or you may use an absolute address. To reference a line number, put a slash in front of the number. LD instructions may also use a slash in front of a number to reference a line in the program.

examples:

| | |
|---|---|
| CALL /1200 | calls the subroutine at line 1200 |
| JR /80 | jumps to line 80 |

CALL H0A2A    calls the ROM routine at 0A2Ah

LD HL,/TB1    where TB1 is defined by a LET statement at the end of the program as some line in the program, HL is loaded with the address of that line.

6. Instructions are exactly as printed in the ZX81/TS1000 manual appendix with two small exceptions:

1. EX AF,AF' should leave out the prime sign (that is, EX AF,AF ).

2. IM 0, IM 1, and IM 2 are written without spaces (that is, IM0, IM1, and IM2)

Note, this is how the disassembler prints them, too.

6a. As an optional shorthand device, the '£' symbol may be used in place of '(HL)'.

7. There are four pseudo-ops:

1. ORG specifies the run-time address for the first instruction in your program. If used more than once, only the last ORG is recognized. If no ORG is given, 16514 is assumed.

2. DATA puts data in memory in sequence with the rest of your code. There is no limit to the number of operand parts, each seperated by a comma. There are three catagories of operand parts allowed:

a. simple constants or variables. If the value is between 0 and 255, it is stored as one byte. If the value is between 256 and 65535 it is stored as two bytes, low byte first. Note, to store H0001 as two bytes you must use H01,H00 .

b. string data. In DATA statements only, any number of characters within quotes is stored in entirety. Same rule concerning no quotes or enters.

c. line number address. You may use a slash in front of a constant or variable to store a line number address, but then make that the only operand part in that DATA statement.

3. RESV puts a specified number of zeros in memory.

ex. RESV 5 is the same as DATA 0,0,0,0,0

4. END is required as the last statement of your source code.

Having entered your source code, call the assembler with a RAND USR xxxxx (depending on version) and then Z. If the assembler finds errors, it will print 'ASSEMBLY ERROR' and a report with an error number and line number. The report codes are in appendix A. You will be left in Basic to edit your mistake; do so, and recall the assembler.

If no errors are found, the output code will be deposited in a string named A$. You will be put in disassembly mode with the top of the page as the first byte of your code and the EOF set to the last. With this, you are set to use the move or save commands. If you move the display, make note of the TOP address so you can find it later.

## Tips

It is considered good programming to be liberal with comments. Especially with this assembler, where line numbers are used over labels for referencing, you have lost a certian readability good comments will restore.

Of course, save your source code before you run your program, for even simple mistakes may crash the system requiring you to reset the machine. If you save your source right after an assembly (with the Basic SAVE), A$ will be saved along with it.

If you are going to run your code in a 1 REM statement (location 16514 and on), you may want to generate the REM before you assemble. Make it at least as big as you expect the code to be (you can estimate high with no ill effects), and change the first character to a semicolon so the assembler will ignore it. Assemble, move the code to 16514 (4082 hex) (TOP and EOF are already set, but check the length), and try your program.

To save your machine code in a 1 REM without the source code, move the code block to a protected part of memory (8 to 16K or above a lowered RAMTOP) or save your code to tape. Execute a NEW, then generate a REM and reload your code into it.

## IV. Sample Programs

```
  1 REM ;XXXXXXXXXXXXXXXXXXXXXXX
  3 REM ;**HORIZANTAL LEFT**
  5 REM ;***** SCROLL ******
  7 REM ;
 10 REM LD B,22;LINE COUNTER
 12 REM LD HL,(DFIL);1ST LINE
 15 REM ;
 20 REM PUSH BC
 22 REM LD BC,31
 24 REM INC HL;1ST POSITION
 26 REM LD A,£;SAVE FOR WRAP
 28 REM PUSH HL
 30 REM POP DE
 32 REM INC HL
 40 REM LDIR ;SCROLL 1 LINE
 50 REM DEC HL
 52 REM LD £,A;FILL LAST POS.
 54 REM INC HL
 60 REM POP BC
 62 REM DJNZ /20 ;LOOP
 65 REM ;
 70 REM RET
 80 REM END
100 LET DFIL=16396
190 GOTO 9999
199 REM
200 REM TEST PROGRAM
205 FAST
210 FOR I=1 TO 300
220 PRINT AT RND*21,RND*31;"*";
230 NEXT I
235 SLOW
240 LET T=USR 16514
250 GOTO 240
```

**This 23 byte program moves everything in the display to the left one position and fills in the last column with the contents of the first. It assumes a fully expanded display file.**

**Note a 1 REM has been generated (and the first character changed to a semi-colon) to move the code after assembly.**

**The test program puts 300 random asterisks on the screen then continually calls our scroller.**

**Replace line 26 with:**

**26 REM SUB A**

**for a non-wrap-around scroll.**

**After assembling, move the code to the 1 REM and call the test program with a GOTO 200.**

```
   1 REM ;XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X
   5 REM ;**BIG CHARACTERS**
   6 REM ;**   PROGRAM   **
   7 REM ;CHAR AT LOC. 16507
   8 REM ;SCREEN POS. ON LINE 34
  19 REM ;
  20 REM LD A,(16507)
  22 REM LD DE,H1E00 ;CODE*8 +
  24 REM LD L,A ; 1E00 =
  26 REM LD H,E ;CHAR. TABLE
  27 REM ADD HL,HL ;POS.
  28 REM ADD HL,HL
  29 REM ADD HL,HL
  30 REM ADD HL,DE
  32 REM LD E,8 ;LINE CTR.
  34 REM LD BC,H080C ;PRINT POS.
  38 REM ;
  40 REM PUSH BC
  42 REM PUSH DE
  44 REM PUSH HL
  46 REM CALL H08F5 ;PRINT AT
  48 REM POP HL
  49 REM ;
  50 REM LD C,£ ;P/U BYTE
  52 REM LD B,8 ;BIT CTR.
  60 REM RL C ;TEST BIT
  62 REM LD A,"█"
  64 REM JR C,/70
  66 REM SUB A
  70 REM RST H10 ; PRINT
  72 REM DJNZ 60
  78 REM ;
  80 REM POP DE
  82 REM POP BC
  83 REM INC HL ; NEXT BYTE
  84 REM INC B ; NEXT LINE
  86 REM DEC E
  88 REM JR NZ,/40
  90 REM RET
 100 REM END
 199 REM
 200 REM TEST PROGRAM
 210 LET A=CODE INKEY$
 220 IF A=0 THEN GOTO 210
 230 POKE 16507,A
 240 RAND USR 16514
 250 GOTO 210
```

Another graphics program, here playing with the character table in the upper 512 bytes of ROM. Code to be displayed is passed through tne unused system byte at 16507. Two ROM routines are called: Print AT and print a character. Two loops are set up to process the 8 lines and the 8 bits per line.

The error report is given similar to Basic errors, that is, error number - slash - line number. These error numbers apply:

1 NON REM STATEMENT or NO END STATEMENT. Only REMs are allowed before the REM END and the REM END is required.

2 INVALID OP CODE. Neither a Zilog mnemonic nor a ZX Assembler pseudo op has been found. Remember, blanks are not allowed before or within the op code.

3 BAD OPERAND SYNTAX. This is the largest category and could be many things. Check for correct number of operand parts, spelling, punctuations, or the use of parenthesis.

4 VARIABLE NOT FOUND. You have specified a variable which can't be found in the Basic variable area. Perhaps you forgot to 'RUN' through your LETs.

5 DATA OUT OF RANGE. You have given a value greater than 255 where 8 bits is required or 65535 where 16 bits is required.

6 PROGRAM TOO LARGE. The assembler has encountered insufficient head room to process the assembly. You must either raise RAMTOP or assemble your program in two or more segments.

7 JUMP DISPLACEMENT TOO LARGE. You have asked for a JR or DJNZ instruction to jump over 128 bytes away. Change to a JP instruction.

8 INVALID LINE NUMBER. You have referenced a line number beyond the end of your program.

The Assembler/Disassembler contains some machine code routines which you might want to use or maybe look at with the disassembler and glean some interesting techniques. They are given here with the address for the hi-mem version. Subtract 4000h to get the low-mem address.

| address | my name | description |
|---|---|---|
| 7FEA | INKY | reads the keyboard returning the character value in the A reg. and the L reg.. No key pressed returns zero, the break key 255, and the shift key 254. |
| 7FDF | INKW | waits for a no key condition, then calls INKY. |
| 7F6C | PRTN | prints the contents of HL. Location 16417 (4021h) is used as a control byte as follows:<br>bit 0 reset - print decimal<br>set - print hexadecimal<br>bit 1 reset - signed decimal<br>set - unsigned decimal<br>bit 7 reset - 4 hex digits<br>set - 2 hex digits |
| 7FD1 | PC1 | prints the character or token of the code in the A reg. |
| 7FB0 | PRTC | prints the characters following the call until a 70h byte is reached. |
| 7FC6 | TAB | tabs the print position to the column specified by the L reg., setting to blanks everything on the way. |
| 7F4A | INPUT | inputs a number to HL, prompting as described for the assembler/disassembler. Location 4021h is used as described for PRTN. A shifted Q will set the carry. |
| 771E | SCRDN | scrolls the screen down and sets the print position at 0,0. |
| 770B | SCRUP | scrolls the screen up and sets the print position at 21,0. |

The table of 26 addresses for each key command starts at 74FC. You may further explore the assembler/disassembler by tracking down those addresses. Unassigned keys are routed to 754E. You may want to change unassigned keys to jump to your own routines.

Here are described many of the useful ROM routines. There are more, but they are often difficult to utilize. For in-depth explainations, consult Dr. Ian Logan's disassemblies.

| address | |
|---|---|
| 0000 | RST 0. Not really useful, but you should know this is the cold start for the system. Clears RAM, sets up a display file, etc. |
| 0008 | RST 8. The error restart. Prints the Basic error message. The byte following this call is incremented and taken as the error number. Line number is taken from PPC (4007h). |
| 0010 | RST 16. Print a character. Code is taken from the A reg. and must be 0-3Fh, 80h-BFh, or 76h, else you will crash. AF, BC, DE and HL are all saved and restored. If bit 1 of FLAGS is set, output will be sent to the printer. |
| 0028 | RST 40. The floating-point calculator. This facinating routine takes up about one-third of the ROM and performs all the Basic arithemetic functions and operations plus a number of special internal ones. The bytes following this call are treated as instructions to manipulate data (five bytes each entry) on the calculator stack. The end of this instruction string is signaled by a 34h byte. Not just numbers, but strings are manipulated by this calculator, their 5 byte entries containing address and length pointers. The list of instruction codes is given at the end of this section. RST 40 does not save and restore most registers. |
| 02BB | Scan keyboard. Returns in HL the 16 bit 'key value'. |
| 031E | Save one byte. Outputs to the cassette port the byte pointed to by HL. Unfortunatly, there is no simple similar routine for loading. |
| 07BD | Decode keyboard. Takes 16 bit 'key value' from BC and returns with HL pointing to the proper character in the tables 007E to 0110. |
| 0869 | Copy screen. Just like the Basic command. |
| 08F5 | Print AT. Enter with line no. in B reg. and column no. in C reg. Computes new DF-CC with error checking. |

| | |
|---|---|
| 099E | 'Make room'. Used by the system to insert lines, variables, expand the display file, etc. Updates the system pointers when through. From (HL), makes BC spaces. CALL 099B makes one space. |
| 09D8 | Finds the address of a line number in a Basic program. Enter with line no. in HL; exit with address in HL. |
| 0A1F | Clears the bottom screen and does a Print AT 23,0. |
| 0A2A | Clears total screen, just like Basic command. |
| 0A60 | Reclaim memory. Used by the system to delete lines, variables, etc. At HL, deletes BC spaces, updating the necessary system piontors. |
| 0A98 | outputs a decimal number, up to 9999, taken from BC reg. |
| 0B6B | prints a string of characters, including tokens, pointed to by DE with length BC. |
| 0BB2 | Plot, unplot. Enter with B reg. holding Y coordinate, C reg. holding X. To plot, poke T-ADDR (4030h) with A0h; to unplot, poke it with 0. |
| 0C0E | Scroll, just like Basic. |
| 0F23 | Fast, just like Basic. |
| 0F2B | Slow, just like Basic. |
| 0F46 | Tests break key and carry is set if not pressed. |
| 0F4B | Debounce keyboard. Call after a keyboard read. |
| 111C | Finds variables in the variable area. To call, load CH_ADD (4016 & 7) with the address of a string of the name of the variable you wish to find. Also, bit 7 of the C reg. must be reset. On return, the carry will be set if no name was found, or the carry will be reset and HL will point to the last letter of the name in the variable area. Increment HL and you may pickup the value of the variable, or if an array or string, its length parameters. |
| 1308 | 16 bit multiply. HL = HL * DE. |
| 149A | Clears the variable area. |
| 1520 | BC to floating point. Converts the number in the BC reg. to a 5 byte f.p. number on the f.p. stack. |
| 158A | F.P. to BC. Opposite of BC to f.p. Carry is set if result is greater than 65535. |

Calculator instructions

| code | |
|---|---|
| 00 | jump if stack top is true, displacement byte follows |
| 01 | exchange top two values |
| 02 | delete top value |
| 03 | subtracts top value from 2nd value. |
| 04 | multiplies top two values |
| 05 | divides 2nd entry by top |
| 06 | 2nd entry to power of 1st |
| 07 | ORs top two numbers |
| 08 | ANDs top two numbers |
| 09 | less than or equal operation |
| 0A | greater than or equal op. |
| 0B | not equal op. |
| 0C | greater than op. |
| 0D | less than op. |
| 0E | equal to op. |
| 0F | addition op. |
| 10 | ANDs a string and a number |
| 11 | string <= op. |
| 12 | string => op. |
| 13 | string <> op. |
| 14 | string > op. |
| 15 | string < op. |
| 16 | string = op. |
| 17 | adds two strings |
| 18 | negates top number |
| 19 | CODE function |
| 1A | VAL function |
| 1B | LEN function |
| 1C | SIN function |
| 1D | COS function |
| 1E | TAN function |
| 1F | ASN function |
| 20 | ACS function |
| 21 | ATN function |
| 22 | LN function |
| 23 | EXP function |
| 24 | INT function |
| 25 | SQR function |
| 26 | SGN function |
| 27 | ABS function |
| 28 | PEEK function |
| 29 | USR function |
| 2A | STR$ function |
| 2B | CHR$ function |

| code | |
|---|---|
| 2C | NOT function |
| 2D | duplicates top entry |
| 2E | divides, returns int. quotient and remainder |
| 2F | unconditional jump, displacement follows |
| 30 | stack a number, data follows |
| 31 | DJNZ (using BERG), disp. follows |
| 32 | less than 0 test |
| 33 | gr. than 0 test |
| 34 | end calculation |
| 35 | used by trig. functions |
| 36 | integer truncation |
| 37 | execute f.p. op. in A reg. |
| 38 | converts E-format to floating-point |
| 86, 88, & 8C | calls series generator for trig. and log. functions |
| A0 | stacks a 0 |
| A1 | stacks a 1 |
| A2 | stacks ½ |
| A3 | stacks pi/2 |
| A4 | stacks 10 decimal |
| C0 through C5 | stores top f.p. entry in system MEMBOT area |
| E0 through E5 | recalls MEMBOT entry to the f.p.stack |

Useful Books Appendix D

It is recomended you purchase two 'Bibles':

1. A manual on the Z80 chip. If possible, go to a bookstore or book department of a computer store and choose one whose style you like. In the field are:

The Z80 Assembly Language Programming Manual and The Z80 Technical Manual by Zilog

Z80 Assembly Language Programming by Lance Leventhal, Osborne/McGraw Hill

How to Program the Z80 by Rodnay Zaks, SYBEX

The last one is available at Radio Shack stores.

2. A manual on the 8K ROM. There is only one and it is good:

Sinclair ZX81 ROM Disassembly Parts A & B by Dr. Ian Logan, Melbourne House

In addition, there are a few books out specifically on the ZX81 and machine code:

Mastering Machine Code on the ZX81 by Toni Baker, Reston

Machine Language Programming Made Simple by Michael Roberts, Melbourne House

Understanding Your ZX81 ROM by Dr. Ian Logan, Melbourne House

Mastering Machine Code is quite good. It explains the Z80 chip fully, but spends less time on this than the other books, instead delving right into many meaty sample programs, ways to save machine code, and even using the floating-point calculator. Programming Made Simple and Understanding the ROM are primarily tutorials on the Z80 chip and its instructions. Made Simple quickly describes only one full-blown sample program (checkers). Understanding uses bits of the ROM as examples, but only has one thin chapter on its overall organization.

V1.0 Notes

In the assembler, JR and DJNZ instructions cannot reference absolute addresses. In fact, the slash is actually optional; they will always reference line nos.

On indexed instructions utilizing a displacement, that displacement may only be a decimal number, not hex. The acceptable range on that number is -128 to +127. However, during disassembly in decimal mode, the displacement is printed as a decimal number 0 to 255, 255 being equivalent to -1, etc.

The instruction in the form LD (XY+d),N (indexed 36h instruction)is mis-disassembled as a three byte instruction instead of four, the third byte being used twice. Examples in the ROM of this bug start at 0644h.